Europäisches Patentamt

European Patent Office

Office européen des brevets

Bescheinigung

Certificate

Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. **Patent application No.** **Demande de brevet n°**

00650094.6

OIPE JC67
NOV 0 2 2001
PATENT & TRADEMARK OFFICE

RECEIVED
NOV 0 7 2001
Technology Center 2100

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

[illegible]

I.L.C. HATTEN-HECKMAN

DEN HAAG, DEN
THE HAGUE, 26/07/01
LA HAYE, LE

EPA/EPO/OEB Form 1014 - 02.91

Europäisches Patentamt | European Patent Office | Office européen des brevets

Blatt 2 der Bescheinigung
Sheet 2 of the certificate
Page 2 de l'attestation

Anmeldung Nr.: Application no.: Demande n°:	00650094.6	Anmeldetag: Date of filing: Date de dépôt:	04/08/00

Anmelder:
Applicant(s):
Demandeur(s):

Mobileaware Technologies Limited
Citywest Campus, Dublin 24
IRELAND

Bezeichnung der Erfindung:
Title of the invention:
Titre de l'invention:

An E-business mobility platform

In Anspruch genommene Prioriät(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat: State: Pays:	Tag: Date: Date:	Aktenzeichen: File no. Numéro de dépôt:

Internationale Patentklassifikation:
International Patent classification:
Classification internationale des brevets:

G06F17/60

Am Anmeldetag benannte Vertragstaaten:
Contracting states designated at date of filing: AT/BE/CH/CY/DE/DK/ES/FI/FR/GB/GR/IE/IT/LI/LU/MC/NL/PT/SE/TR
Etats contractants désignés lors du depôt:

Bemerkungen:
Remarks:
Remarques:

"An E-business Mobility Platform"

Introduction

The invention relates to provision of applications and content to end users.

In recent years there has been a rapid development of technologies both for the end user to receive content and applications and for hosting of the content and applications. At the user end a wide range of devices including PCs, handheld computers, WAP phones, PDAs or digital TVs may be used. Content providers and corporate systems also use a number of different technologies including servers communicating using HTML, XHTML, and other XML compliant languages.

It is believed that there is a fundamental paradigm shift from software product delivery to software services delivery. This paradigm shift has been growing over the last decade and is now achieving critical mass whereby the information society will comprise an ubiquitous array of thin-client computing devices that are always connected to centralised server-based application services. MobileAware believe that this shift will include a wireless computing device, hence referred to as a mobile terminal. MobileAware believe that mobile devices will become the primary driver of this paradigm shift as the mobile device is the ultimate manifestation of thin client and thin client is the ultimate device for enabling the "Software as a Service" paradigm.

Current wireless technology is developing in a direction that is parallel but not convergent with this paradigm transition. In fact, technologies such as WAP have focused mobility at the protocol level and have built an environment parallel to Internet standards. This is insufficient to address the requirements of mobile users as it does not enable usability advantages at the e-business application and Web authoring levels.

It is therefore an object of the invention to provide an e-business mobility platform to achieve such integration in an effective and versatile manner.

Other objects are to:

- allow content providers to easily provide personalised application and content in terms of behaviour and presentation to users on the appropriate devices without the need to rewrite and maintain content in different formats,
- allow company staff to have a single secure and personalised point of access to all distributed application services, regardless of device used,
- allow application service providers (ASPs) to make their applications available to users via any device and to manage different communities and share applications among groups in an efficient, cost effective, and secure manner, and
- allow operators to package applications for personalised delivery to end-users according to their lifestyle.

Statements of Invention

According to the invention, there is provided an E-business mobility platform comprising:-

an end user interface comprising means for interfacing with end users having different user devices;

a server interface comprising means for interfacing with a plurality of servers each providing content or applications; and

a session controller comprises means for controlling bi-directional session communication between an end user and a server, between one end user and another end user, and between a server and another server.

In one embodiment, the session controller comprises means for creating a mobility session object associated with each session, and means for creating event objects associated with the mobility session object and representing requests that occur within a session.

In one embodiment, the mobility session object contains the variable properties that represent the state of an end user session including the current network communication channel in use and the location of the end user.

In one embodiment, the session controller comprises means for allowing hosted services to interrogate the mobility session object independent of their physical location.

In one embodiment, the session controller comprises an event bus, means for placing an object on the event bus, and a plurality of modules each having publish and subscribe access to the event bus.

In one embodiment, the session controller comprises a policy engine comprising means for activating and executing a policy associated with a session.

In one embodiment, the policy engine comprises means for mapping an event object to a policy.

In one embodiment, each policy is a decision tree.

In one embodiment, the session controller comprises means for generating a policy context, including end user data, including user device parameters and user requirements.

In one embodiment, the session controller comprises a service selector module, comprising means for selecting an appropriate service based on a request.

In one embodiment, the service selector module comprises means for calling a policy associated with a selected service, and said policy comprises means for controlling access to the hosted service.

In one embodiment, a hosted service comprises means for generating an event.

In one embodiment, the session controller comprises service request handlers, and each policy comprises means for activating said handlers to access a hosted service.

In one embodiment, the session controller comprises means for using a task mark-up language that provides high-level presentation instructions that control how device-neutral content is transformed to device-specific formats.

In one embodiment, the session controller comprises a content assembly module comprising means for parsing server responses and generating a representation based on mobility session object attributes and transforming the input content.

In one embodiment, the session controller comprises a delivery module comprising means for reading server responses from memory and for placing them on a dynamically selected output channel for the end user.

In one embodiment, the session controller comprises means for creating a composite response of more than one page, associated with the original user-requested task.

In one embodiment, the capabilities of a hosted service can be represented as a set of decisions that can be utilised by a policy.

Detailed Description of the Invention

The invention will be more clearly understood from the following description of some embodiments thereof, given by way of example only with reference to the accompanying drawings in which:-

Fig. 1 is an overview schematic representation of an E-business mobility platform;

Fig. 2 is a diagram illustrating interactions with a Policy Engine of the platform; and

Fig. 3 is a diagram illustrating a policy.

Referring to Fig. 1 an E-business mobility platform 1 is illustrated. The platform 1 comprises, at the hardware level, an EJB application server cluster. At the software level, the platform 1 is modular and modules may be added and removed at run-time as required. Module feature definitions are registered with an administration module within a "quadrant" of core functions. At a high level the functionality falls within the following categories: (a) an end user interface , (b) a server interface , and a session controller in-between.

The following describes the structure of the modules.

Orthogonal Methods

Core module features are mutually orthogonal, insofar as the features offered by any one module should not be replicated by the combination of one or more features from other modules.

A module may contain one or more implementations of a specific feature (e.g. Secure Certificate Generation). Where this occurs, the module may be informed which implementation is intended during an invocation, or alternatively the module may be permitted to select its default. Multiple service implementations do not break the requirement of modular orthogonality, so long as the multiply implemented features can only be accessed via the same module.

Referring to Fig. 2, an Event Bus (also shown in the central circle of Fig. 1) provides an event stream to (instances of) a Policy Engine. A policy engine invokes methods on modules and receives results, which influence the action of the policy engine. Event notifications are transmitted via the Event Bus. Details of methods are registered with a Registry of the core 2. This data is used by the Policy Engine when invoking Questions and Actions (registered methods taking zero or more parameters) of registered modules.

Categories

Module features fall into several categories:

Question.	A Question feature is used in a policy to acquire information from the module, information that will subsequently be used to guide the policy.
Action.	An Action feature is used in a policy to commence some activity because of a decision taken in the policy.

Trigger. A Trigger feature is a source of events (represented as event objects) that are passed to the Event Bus for subsequent delivery to modules that have registered interest in specific events. The Policy Engine is the principal destination for such events, where they are used to activate policies.

Admin. An Admin feature is used by an Administration module to control and monitor other modules.

Module Structure

Modules may be implemented as one, two or three parts:
A one-part module must implement the Core API and all of the features of the module.

In a two-part module, the first part provides an API encapsulating the necessary and sufficient methods of the associated service. The second part provides the implementation of these methods. Typically, the second part is a simplified, locally developed version of the service.

A three-part module is a modified two-part module where a Service Adaptor and a Service Implementation have replaced the implementation part. The Service Implementation is typically a third-party (OEM'ed) service exposing a standard or proprietary API.

A "Hosted Service" is a one, two or three-part module representing a specific service. A hosted service may provide content intended for the end user, or provide access to an application that generates a result intended for the end user or intended for use by other hosted services. A one part hosted service is co-located with the server. A two part hosted service uses a core module acting as a proxy or adapter for an existing service that may be co-located or externally accessed. A three part hosted service uses an intermediate adaptor to translate between the API used by the core system and the API used by the external service.

As an illustration, a food delivery business provides a menu based selection facility, and delivery charges based on distance. This service is accessed as a hosted service. Furthermore, another hosted service offering location information is employed by the food delivery service to determine distance and assess the delivery charges. The order in which these services are used to generate a response for the end user is determined by a policy associated with the service.

APIs

There are several APIs in the platform. These are either a specific API with a fixed definition or a member of a class of APIs with common features. The details are given below:

Core API. (Specific). The interface between the Core 2 and any modules that are present or may be added.

Adaptor API (Class). Such APIs offer a generic feature set of a specific service and are implemented by Service Adaptors to connect to corresponding Service Modules.

Service API (Class). Such APIs offer a specific feature set of a specific service and are implemented by Service Adaptors to gain access to the service. The API may be defined by a third-party service and must be capable of connecting to the core APIs, preferably using the same implementation language as the core 2.

Admin API (Specific). This API is used to facilitate the provision of a user-oriented interface to the Administration module. The Admin module uses this API to support one or more user-interfaces, possibly concurrently, that are used to administer the platform.

Request API (Class). This API is used by the Content Request Handler Module to forward requests for content to the content providers. Providers may use HTTP or other protocols as carriers for requests for content. Provider Proxy sub-modules implement the Request API and the API of the provider to facilitate content requests to specific content providers.

Generic Module API. Associated with each module in the system is a generic module API. This is required to communicate with the Core 2 and the Policy Engine. The calls are set out in the table below.

Generic Module API Calls			
Name	Parameters	Return Type	Description.
Ask Question	Question Id Parameter values	Integer	Tells a module to process the Question with given Id. Once the module has determined the result, it returns the index into the result list previously registered with the Policy Module.
Perform Action	Action Id Parameter values	<none>	Tells a module to process the Action with given Id. This call blocks the calling thread until the Action is complete and then returns.
List Questions		List of Questions	List all Questions that this module is capable of answering
List Actions		List of Actions	List all Actions that this module is capable of performing
Get Module		String	Gets the unique string identifying this module. E.g.

Name			"com.mobileAware.Everix.ContentAssembly "

Databases

There are several databases within the platform 1. All of these databases are hosted within the same database management system, and may be accessed directly or via entity objects that represent the data. Security of this data rests with a security mechanism provided by a database management system (DBMS). The databases are:
Content Storage DB. Holds content in a structured manner for retrieval by the Content Assembly Engine.

Content Transition DB. Holds original and assembled content prior to delivery. Content is held here until actively expired. This DB can be used to hold pre-emptive content (that has been assembled in anticipation of being required) and thereby improving the perceived response time.

Profile Agent DB. Holds end-user preferences, end-user statistics, settings and any other data pertaining to the end user or user groups. It also holds session handles on behalf of the end-user.

System Configuration DB. Holds all data necessary to boot the system and set it up for normal running. To boot the system, the bootstrap process only requires the means of accessing this database.

Policy

A policy is a set of rules whose order of execution is dictated by the results of queries within the rules. The most natural view of a policy is a tree. The root of the tree is the initial Action or query (Question). From the root follow more Actions and Questions, taking different paths according to the results obtained from queries.

An Action is a special sort of query, distinguishable by the fact that it has only one result (i.e. success).

A Question can return one of two or more possible results. Internally, the result is an integer within a defined range, where each possible value represents a different possible result and it is normal to associate these values with phrases corresponding to these results. For example, the numbers 1 and 2 could represent "yes" and "no" respectively.

In the running of a policy an Action or Question may unexpectedly fail. In the case of such a failure, a policy must abandon further execution in favour of an alternative line of execution. The alternative line (which is an alternative policy) will handle the cause of the failure. Each policy may be associated with an exception handling policy that is executed in the event of an exception being detected by the executing policy. If no handling policy is defined, the policy will use the handler of its calling policy (if any) or throw the exception to the container (if no handler is defined).

An Action or Question is executed in the module that has registered the Action or Question in the Registry. When the policy designer indicates an Action or Question in a policy by name, a mapping from the name to the actual module is used by the Policy Engine to determine how to call the required method. The Policy Engine passes the name of the Action or Question to the module, together with any required parameters, and awaits a response from the module.

The Policy Engine is the mechanism that executes policies. Every registered event type is associated with a particular policy, which commences execution with the Policy Engine whenever the event is observed. When a policy is started in this way, a context is created for the executing policy and the details of the event are stored in the context. The policy designer determines which event or events can trigger the policy. This association is stored in a Policy/Event Map.

The Policy Engine may also start a policy by being instructed to do so, rather than as a consequence of observing an event. Any module that requests the Policy Engine to execute a policy directly must also provide a policy context ID (from a running policy). The Policy Engine does not create a new policy context in this case.

A policy may contain a reference to another policy. If this reference occurs at the leaf of the policy tree, then this is equivalent to a "jump" to another tree. The referenced policy tree takes over and inherits the original policy context. If the reference occurred as a non-leaf of the policy tree, then this is equivalent to a "call" to another tree. Again, the referenced policy tree inherits the original policy context. When the referenced policy tree reaches a leaf (the end of a policy) it returns to the calling policy.

It is possible to create mutual references within a set of policies that could lead to infinite execution. This should be avoided through manual or automated inspection at the policy authoring stage, before making a policy available for execution.

Since policy trees should not be organised into loops, if a policy requires iteration then this should be performed within a separate module. If the loop requires execution of policies, then the loop can invoke the policies directly, and using the current policy context. It is beyond the scope of the policy editor to prevent infinite loops caused by modules.

Data generated or required by Actions or Questions may be stored in, or retrieved from the policy context and/or a session object.

Fig. 3 illustrates a policy entry point, which is followed by a Question called "invokeService" provided by the SEC module. The result of the Question will determine which of the three possible branches will be followed. The branch marked "Failed" leads to an Action called "report" provided by the "ER" module. The sample also illustrates calling another policy, in this case the "sendTML" policy.

List of modules

Referring to Fig. 2, the Registry stores all the details of registered modules, the methods they support, the events they can generate, access restrictions that apply and any other information required at run-time. The contents of the Registry is also used by the Policy Manager to get information about the available modules and support the policy editor GUI.

Every module must be listed in the Registry before it can appear in the Policy Manager, and subsequently make its features available to the rest of the system. The Registry provides a means of enumerating all the registered modules.

List of Questions/Actions

For each registered module, the Registry maintains two lists of methods that the module provides.

One is a list of Question methods. These methods take zero or more parameters and return one value from a set of possible values. The set is known in advance, so the result will actually be expressed as an index into a list representing the results. For the policy designer, indices are not sufficiently expressive, so the Registry will also hold textual versions of these results (as provided by the module during registration).

List of Event types

Every module may create events and place them onto the Event Bus. Events are categorised according to the source module and module-assigned name. The Registry maintains details of the event types, such as a textual summary.

Event Bus

The Event Bus is the principal means of communication in the system. It is an efficient real-time internal distributed notification system that alerts the various components to changing situations. The Bus carries objects representing events (usually associated with sessions). Larger data can be conveyed using the mailbox approach, where data is held by the Session Manager while the Event Bus carries a notification that the data is there to be read.

Publish/Subscribe (via JMS)

Java Messaging Service supports a variety of message types over a variety of communication models, including Publish/Subscribe. The Event Bus is a core component that uses the publish/subscribe communication model to deliver event objects to any destination that has registered an interest in receiving the events.

Objects

An event object may be created by any registered module. A module may only create events that it has registered. An event can only be registered by a single module. An event carries additional data such as the session ID of the session in which the event occurred (held as a Message Property).

There is a mobility session object associated with each session. Event objects are associated with the mobility session object and represent requests that occur within a session. The session object has variable properties that represent the state of an end user session including the current network communication channel in use and the location of the end user. This is very important as it allows the platform to handle multiple channels for a single end user session. Thus, for example, the end user can switch between a mobile handset and an interactive TV. Also, hosted services can interrogate the mobility session object independent of their physical location, again providing versatility.

A mobility session object represents the variable data associated with a dialogue between the server and the client, including details of the communication channel(s) and device(s) currently used by the client. When a request is received from a client, the details of the request are packaged in an event object and passed to the Event Bus. On observing an event object on the Event Bus, the Policy Manager uses a map in the Registry to identify an appropriate Policy to handle the event and creates a policy context for storage of data during the execution of the Policy. The Policy Engine then executes the Policy, starting by extracting information from the event object. If the event object represents the first request in a dialogue, a new mobility session object will be created, otherwise an existing mobility session object is located using a session key in the event object. In this way, a sequence of requests in a dialogue with the client will be associated with a single mobility session object that can carry information from request to request.

Access control

It is important that the ability to create events and view events is controlled. This includes the ability to view and manipulate event usage in policies. For this reason, events are subject to the Access Control List of the underlying container.

Content Handling

Content is handled in several stages, as detailed below.

Content Preparation

Content may be received in an open format such as HTML or in a platform-specific format, namely Task Markup Language (TML).
If the input format is that required by the end user this format is used throughout.
If the input format is in TML the platform uses the TML representation together with attributes of the current session for internal processing and conversion to a format suitable for the end user format (based on channel and device requirements).
If the input format is not in TML and does not match the end user format then a conversion takes place if a mapping to TML exists. A mapping is defined as a translation from the input's schema to TML.

It is envisaged that the TML grouping mechanisms will support both a visual representation suitable for human interpretation and a non-visual representation suitable for interpretation by automated systems.

TML describes the raw content inherent in a number of formats including HTML, plain text, heterogeneous business objects etc. TML imposes structure on content thereby encapsulating the details necessary to select and arrange the content based on various parameters. This stage is applied when the content may need translation to another representation. This stage is bypassed when the provider format is guaranteed to match the end-user delivery format. When applied, the TML produces a richly structured, loss-less representation of the original content. Inference rules may be applied to give additional structure to the content.

Application of TML to the original content is performed by a number of semi- and fully-automated mechanisms. Where a good structure already exists within the original content, a mapping to TML can be applied automatically at this stage or at a later stage in the content module

Content Acquisition

Two forms of content acquisition are supported:

Content Feed. A feed is a fixed source of input into the system, requiring no specific requests from the system. A news stream is an example of a content feed. This provides an input to a notification-hosted service that automatically transmits notifications to end users according to a preset profile. The profile may reside in the Registry and/or in the hosted services itself.

Solicited Content. Solicited content is provided in response to a request from the system. This form of acquisition occurs when the content is held exclusively outside the system, and in the context of a transaction between the external source and the system (or end-user via the system). Solicited content may be obtained through a Provider Adapter that implements the Request API and the API of the associated content provider.

Content Assembly

A Content Assembly Engine (CAE) receives requests (normally from a running policy) to retrieve data from storage and to reformat it into a form suitable for the characteristics of a particular end-user device. The CAE chooses an appropriate translation processor based on the current mobility features, i.e. user, device and service parameters. The result of this is that relevant content portions are extracted from the enhanced mobility representation of the TML document. Which translator processes are executed, and which data is used as input, is determined by the CAE. The end result is a content object that can be passed to a suitable delivery vehicle.

Delivery

Pre-formatted content may be delivered via any of a number of channels. The channel is selected by the executing policy – which also determines the assembly actions. In some cases, the final rendering of the content requires additional processing and is performed by a client-side applet or script. In most cases, the content can be delivered directly via the appropriate channel. The channel to use is determined either by the running policy, or by context data held in the Profile Agent that represents the end-user.

Delivery Module

The Delivery Module supports an Action method that takes a reference to assembled content, and a session ID, and delivers the content to the appropriate device indicated in the session context. Additional disambiguating information can also be provided in the case of a session being conducted over multiple devices. For example, a user with a PDA connected to a WAP phone could receive a text message on either device so the delivery system needs to know which to select as the target.

Security Framework

The product exposes external security requirements on all product actors. In addition the product itself offer security service capability.
In the context to the design these two security functions impose security design requirements on the product design.

Security

A Security Module (SEC) provides security facilities for communications between the server and external actors (e.g. end-users). It does not provide internal security, which is best handled by a server-side security solution.

Among the features offered by the Security Module are: user authentication, certificate creation, certificate verification, digital signatures, and message encryption. The Security Module provides its features through a separate security solution from a third party, integrated as Hosted Services.

Additional Hosted Service Modules

Corporates can offer their business logic as a module, like any other module in the system. The corporate user would also have to provide a set of policies to handle events related to the service, and any other information such as appropriate access rights. Once in place, the corporate can use the user group management facility of the administration GUI to add users (customers), create communities (of customers), assign default preference values for user/customer profiles and other such configurations.

Location

The Location Module is a placeholder for a real location module. The Location Module provides answers to simple location-based questions such as: "Where is the client?" In order to answer such a question, the Location Module may generate a message asking the user for the answer. In a real Location Module, this question might be answered automatically via a GPS system.

Billing

A Billing Module collects event details and derives Customer Data Records from these details. The records are streamed to a log. There are features for capturing the current content of the log, and for removing captured information from the log, all at run-time without pausing. It is assumed that the records shall be processed externally for the purpose of billing.

Operation of the platform 1 is now described with reference to Fig. 1. The platform 1 essentially acts as a broker in which parties belonging to ASP, content provider, network operator, and the corporate sectors interact with each other and with end users regardless of the communication and file formats used. Thus, for example, a user may order a pizza, the pizza restaurant may use the platform 1 to track location of delivery personnel in real time, and it may also use the platform 1 to communicate with service providers who print maps and provide routing information.

An incoming request from an end user is directed to the Request Handler. This module monitors messages to determine data such as the user identification. The Request Handler uses this data to create a mobility session object and associated Event Objects and these are is placed on the Event Bus, accessed and used by the cluster of hardware machines. The various modules in the platform use publish and subscribe (Java Messaging) functions to monitor the Event Bus. One such module is the Policy Engine, which automatically maps the Event Object to a policy (decision tree). There is one policy per decision tree.

The Session Manager (SM) and the User & Group Profiles (UGP) core modules may be regarded as persistent memory of the platform 1 from the perspective of how an event is handled.

The policy writes a policy context to the SM & UGP, including data such as the user session status or user's mobile number. This data is important as it allows the platform to handle a wide range of different user devices and associated protocols, as set out in Fig. 1.

The Policy Engine calls a Service Selector (SS) according to the context data. The SS calls a policy associated with the service that generates the response to the user's request. If security is required the Security Module (SEC) is called through the policy. For example: based on a request URL (where URLs are being used) the SS determines that the user is seeking the Restaurant Booking service, which maps to the RestBook policy, which in turn is called by the SS. The RestBook policy then interacts with the RestBook hosted service (via Questions and Actions) to obtain the response for the user (expressed in TML).

The results of the Questions and Actions called by the policy are posted to the SM. The service requests are handled by interaction between the Policy Engine and the Hosted Service Request Handlers (HSRH) via a Question/Action interface. The HSRH operates as follows: When a policy has determined which of the hosted services should respond to the client request, the main policy calls the invocation policy of the identified service via the SS. The invocation policy may then use Questions or Actions supplied by the hosted service to execute processes associated with the service. Generally, these processes will retrieve a response expressed in TML that will be deposited in the SM for subsequent assembly by the CAE and delivery to the client.

Fig 1, for clarity, only illustrates Content Provider links, however, the platform 1 hosts a wide range of services including ASPs, corporate systems, and network operators. However, the mechanism for access to these services is as described and illustrated for Content Providers.

The Parse and Store module and the Content Assembly (CA) module receive content in TML (Task Markup Language). They convert to a suitable format for the user according to the event policy. For example, the user's device format may be HTML, WML, SMS, SMTP, fax or VOXML.

The Policy Engine calls the Delivery Module which reads the response and places it on the user's output channel.

Another aspect of TML is that it may produce a composite response comprised of more than one sub-response (or page), and will make elements of this composite response available to the client on demand (starting by default with the first element). This form of response is used where the original response from the hosted service is inappropriately large for the client's browsing device. The CAE may add additional inter-page links to enable the client navigate through the set of pages derived from the original response.

It will be appreciated that the invention provides a platform that achieves the objectives set out in the introductory part of this specification. It provides excellent flexibility for connection of end users to servers. For example, it allows ASPs to provide software as a service on per-user basis with flexible licensing and billing arrangements. For example, the Real Billing Service of the core functions allows flexible billing and licensing arrangement to be made. The platform effectively allows users to fully control their interaction with e-services through a fully personalised interface. The settings for personalisation include function, location, time, device, and other system and application-specific settings from which the user can choose. The platform 1 also allows ASPs to host multiple corporate services and applications on the same server. These can be grouped and shared by different user groups. Also, it will be appreciated that the security module provides advanced industry-standard authentication and authorisation for control of access to applications and data. Also, because of the clustered multi-server hardware architecture, the platform 1 is highly scaleable and there is excellent reliability because of redundant clustering.

At a more fundamental level, the platform provides solutions to the three primary problems concerned with mobility personalisation: (a) Presentation personalisation, (b) Behaviour personalisation, and (c) Task personalisation. This is set out in more detail below.

Mobility Personalisation	
Presentation	Presentation personalisation uses the device modelling and multi-channel publishing mechanisms to categorise devices and map said devices to a set of indirect properties. The Multi Channel Publisher (MCP) provides automated functions to transform XML and HTML content to the target device through the indirect properties. • The device modelling is based on an innovative device hierarchy tree. • The MCP combines the best of automated translation with a high-level intentional mark-up language compliant to XML. This mark-up can be used in standard web authoring processes that enable web content to be structured for multi-device translation without recourse to producing device-specific mark-up.
Behaviour	Behavioural personalisation is achieved through the policy engine and user group preference module which allows requests and responses from hosted services to be intercepted, apply mobility logic, in order to personalise the request or response to the mobility requirements. The policy engine is capable of modelling the notification, time, device location preferences of the end user and extending standard actions with such requirements. The behaviour personalisation can be applied to a user or a group of users.
Task	Task personalisation is focused on a scenario whereby

	the Internet will be populated by many service providers who provide specific vertical services to end-users (e.g. booking a flight). The policy engine and hosted services model allow such service providers to dynamically export their capabilities through a meta-API that allows personalised tasks to be built up by end users through the combination of features from multiple e-service providers. The policy engine also allows such processes to be tracked and reported.

Another advantage is the fact that the business logic of the server interface enables tasks to be aggregated to deliver extended functionality through intelligent agent-based services. The device-centric modelling capabilities employed by the platform 1 enable publishing of a user-personalised interface services to different devices.

The invention is not limited to the embodiments described but may be varied in construction and detail.

CLAIMS

1. An E-business mobility platform comprising:-

an end user interface comprising means for interfacing with end users having different user devices;

a server interface comprising means for interfacing with a plurality of servers each providing content or applications; and

a session controller comprises means for controlling bi-directional session communication between an end user and a server, between one end user and another end user, and between a server and another server.

2. An E-business mobility platform as claimed in claim 1, wherein the session controller comprises means for creating a mobility session object associated with each session, and means for creating event objects associated with the mobility session object and representing requests that occur within a session.

3. An E-business mobility platform as claimed in claim 2, wherein the mobility session object contains the variable properties that represent the state of an end user session including the current network communication channel in use and the location of the end user.

4. An E-business mobility platform as claimed in any preceding claim, wherein the session controller comprises means for allowing hosted services to interrogate the mobility session object independent of their physical location.

5. An E-business mobility platform as claimed in any preceding claim, wherein the session controller comprises an event bus, means for placing an object on the event bus, and a plurality of modules each having publish and subscribe access to the event bus.

6. An E-business mobility platform as claimed in any preceding claim, wherein the session controller comprises a policy engine comprising means for activating and executing a policy associated with a session.

7. An E-business mobility platform as claimed in claim 6, wherein the policy engine comprises means for mapping an event object to a policy.

8. An E-business mobility platform as claimed in claim 4 or 5, wherein each policy is a decision tree.

9. An E-business mobility platform as claimed in any preceding claim, wherein the session controller comprises means for generating a policy context, including end user data, including user device parameters and user requirements.

10. An E-business mobility platform as claimed in any preceding claim, wherein the session controller comprises a service selector module, comprising means for selecting an appropriate service based on a request.

11. An E-business mobility platform as claimed in claim 10, wherein the service selector module comprises means for calling a policy associated with a selected service, and said policy comprises means for controlling access to the hosted service.

12. An E-business mobility platform as claimed in claims 2 to 11, wherein a hosted service comprises means for generating an event.

13. An E-business mobility platform as claimed in claim 12, wherein the session controller comprises service request handlers, and each policy comprises means for activating said handlers to access a hosted service.

14. An E-business mobility platform as claimed in any preceding claim, wherein the session controller comprises means for using a task mark-up language that provides high-level presentation instructions that control how device-neutral content is transformed to specific device formats.

15. An E-business mobility platform as claimed in claim 14, wherein the session controller comprises a content assembly module comprising means for parsing server responses and generating a representation based on mobility session object attributes and transforming the input content.

16. An E-business mobility platform as claimed in any preceding claim, wherein the session controller comprises a delivery module comprising means for reading server responses from memory and for placing them on a dynamically selected output channel for the end user.

17. An E-business mobility platform as claimed in any of claims 14 to 16, wherein the session controller comprises means for creating a composite response of more than one page, associated with the original user-requested task.

18. An E-business mobility platform as claimed in any preceding claim, wherein the capabilities of a hosted service can be represented as a set of decisions that can be utilised by a policy.

ABSTRACT

"An E-business Mobility Platform"

A platform (1) allows end users to engage in a session with servers such as content providers, corporate systems, or ASPs. There is a session object associated with each session, attributes of which include the channel – thereby allowing changing of a channel during a session. Event objects are associated with the session object as requests arise, and an event bus is used for communication. Hosted services may be physically resident on the platform or may be accessed transparently via interfaces. A policy engine activates and executes policies for sessions, each policy comprising a decision tree.

1/3

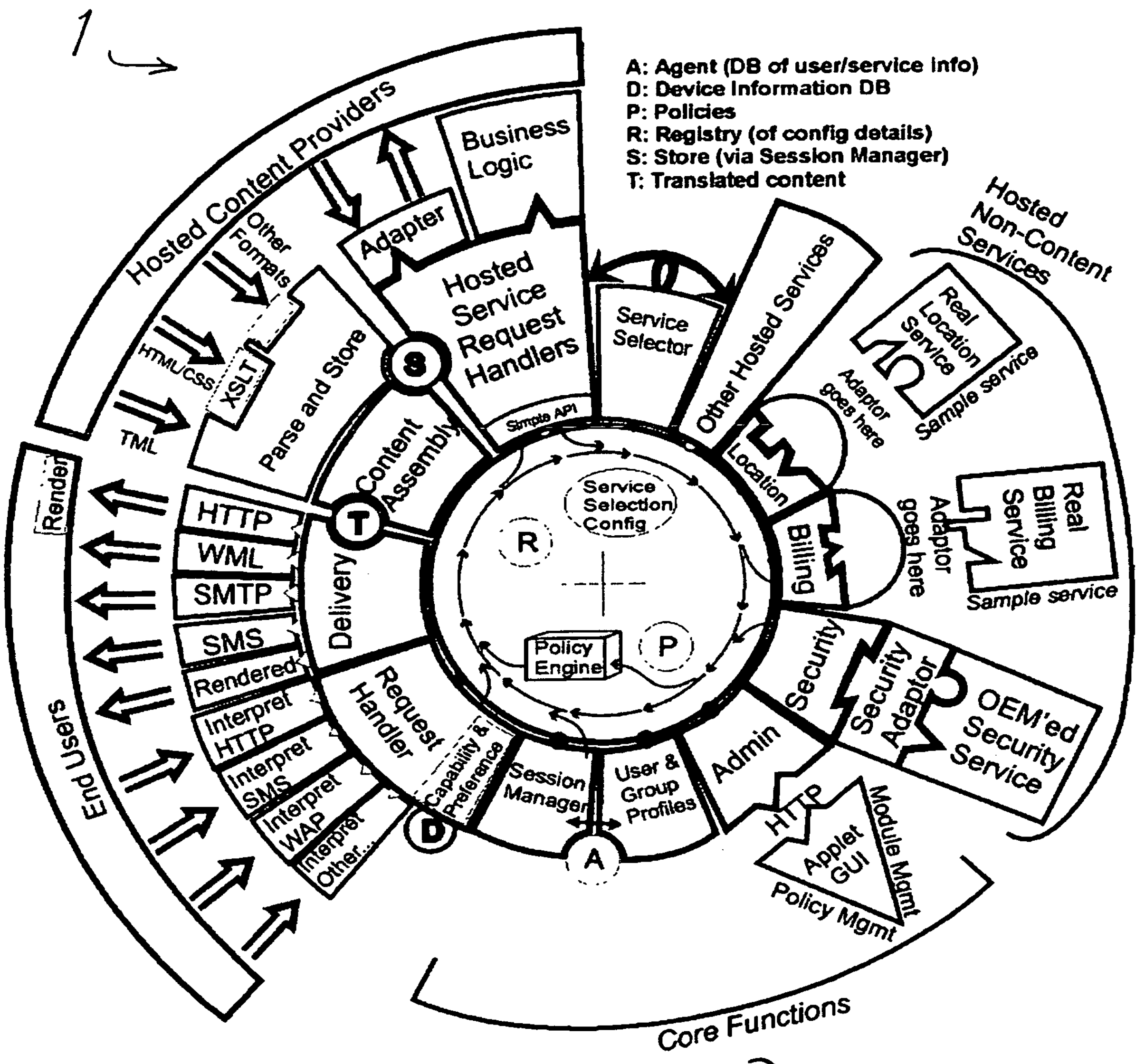
1
A: Agent (DB of user/service info)
D: Device Information DB
P: Policies
R: Registry (of config details)
S: Store (via Session Manager)
T: Translated content
Hosted Content Providers
Other Formats
HTML/CSS
TML
XSLT
Parse and Store
Adapter
Business Logic
Hosted Service Request Handlers
Simple API
Content Assembly
Service Selector
Other Hosted Services
Hosted Non-Content Services
Real Location Service
Sample service
Adaptor goes here
Location
Real Billing Service
Adaptor goes here
Sample service
Billing
Security
Security Adaptor
OEM'ed Security Service
Admin
HTTP
Applet GUI
Module Mgmt
Policy Mgmt
User & Group Profiles
Session Manager
Capability & Preference
Request Handler
Delivery
Service Selection Config
Policy Engine
R
P
S
T
D
A
Render
HTTP
WML
SMTP
SMS
Rendered
Interpret HTTP
Interpret SMS
Interpret WAP
Interpret Other...
End Users
Core Functions
2

Fig. 1

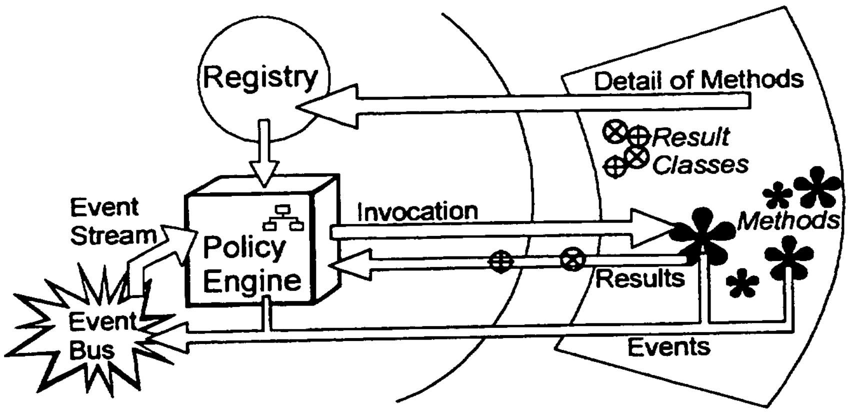
Registry
Detail of Methods
Result Classes
Methods
Event Stream
Policy Engine
Invocation
Results
Event Bus
Events

Fig. 2

3/3

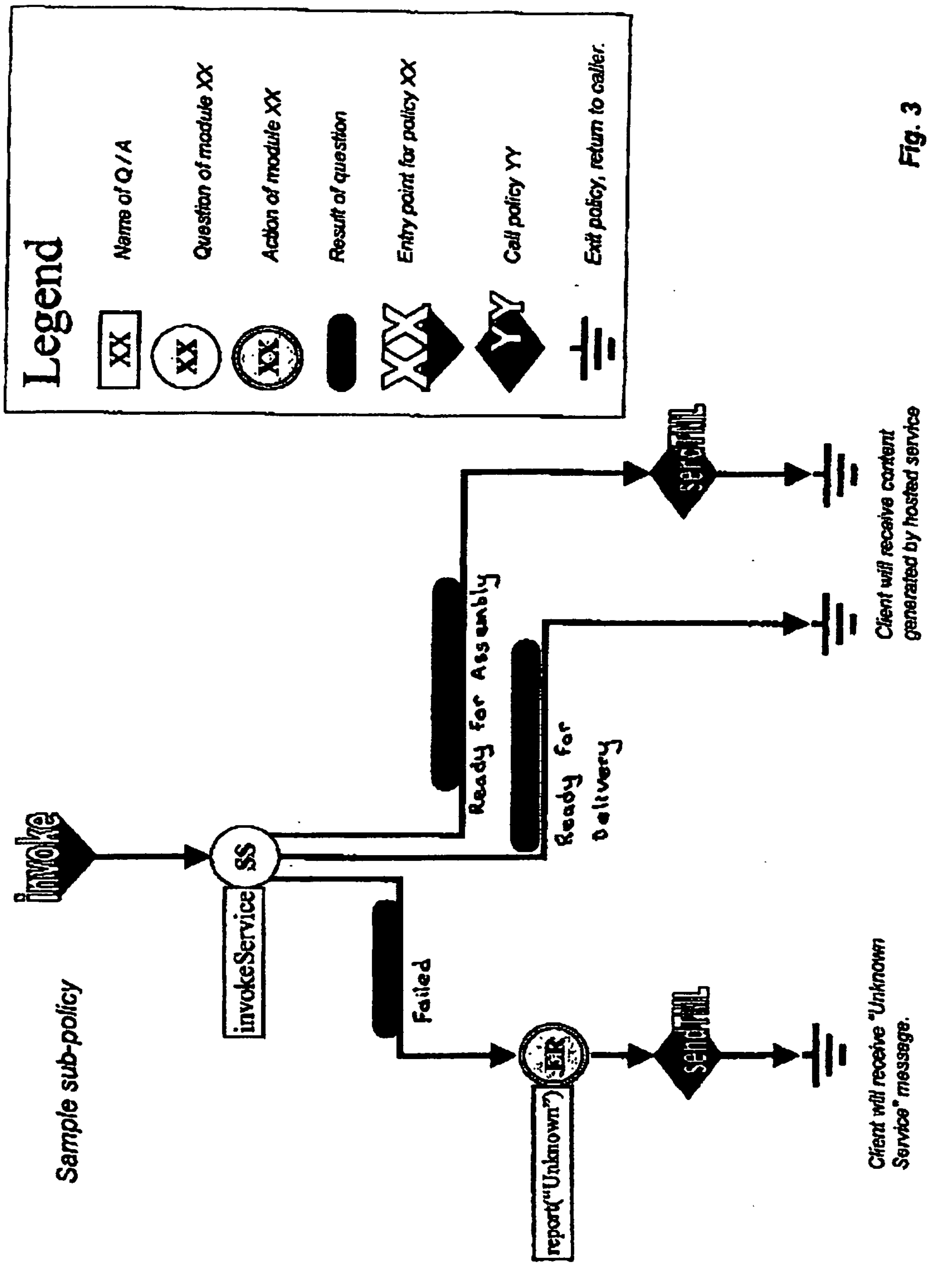
Sample sub-policy
invoke
SS
invokeService
Ready for Assembly
Ready for Delivery
Failed
ER
report("Unknown")
sendHTML
sendHTML
Client will receive content generated by hosted service
Client will receive "Unknown Service" message.
Legend
XX
Name of Q / A
XX
Question of module XX
XX
Action of module XX
Result of question
XX
Entry point for policy XX
YY
Call policy YY
Exit policy, return to caller.

Fig. 3